The Potential Impact of VHDL on Dependable Distributed System Design

M. C. McElvany*
Aerospace Technology Center
Allied-Signal Aerospace Company
Columbia, MD 21045
michelle@batc.allied.com

1991

Abstract

The goal of dependable systems and components is to provide proper service in the presence of faults. One design approach is to develop extremely dependable building blocks, using fault avoidance techniques to decrease generic or design faults. Then, the reliable system components are combined using fault tolerance to create a dependable distributed system.

The adoption and implementation of a standard hardware description language in VHDL (VHSIC hardware description language) promotes the use of fault avoidance techniques throughout the system design cycle. Descriptive languages such as VHDL also provide a basis for tools which can aid in the specification, design, and validation of both dependable systems and their component building blocks. However, integration of VHDL and existing high-level system evaluation tools remains a research problem.

1 Introduction

The success of the Advanced Tactical Fighter (ATF), the Integrated Airframe/Propulsion Control System Architecture (IAPSA), the PAVE PILLAR architecture, and similar high-reliability, high-performance systems requires the development of dependable complex real-time computing systems. The dependability and high-speed performance requirements of life-critical systems cannot be achieved merely by using distributed processing and faster processors; dependability requires the systems to tolerate faults.[1] Consequently, most real-time control systems to date use dedicated computing systems which cannot support other applications.

*Supported by ONR Contract # N00014-91-C-0014

[1] A *fault* is an anomalous physical condition, the identified or hypothesized cause of an *error*, which may eventually lead to a *failure*, a loss of service. Note that a single fault can cause many errors.[1]

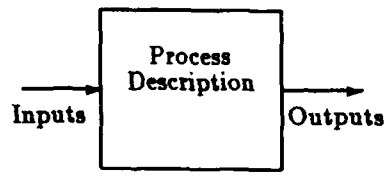

Figure 1: Discrete Process Model

An alternative approach is to develop extremely dependable building blocks which, when combined using fault tolerant techniques, can meet the needs of a particular application with the desired level of reliability. The existing formal theory of dependable system design was derived from specific architectures which discovered and solved many of the problems associated with guaranteeing proper system service in the presence of faults. These architectures include SIFT (Software Implemented Fault Tolerance)[2, 3], AIPS (Advanced Information Processing System)[4], FTPP (Fault Tolerant Parallel Processor)[5, 6], and MAFT (Multicomputer Architecture for Fault Tolerance) [7, 8], a strong candidate for the next generation space shuttle engine controller. [9]

One way to specify proper service is to use a textual or behavioral description of the system or component functions. Using the discrete process model depicted in Figure 1, we can define the types of inputs, the types of outputs, and provide behavioral, structural and data-flow definitions of the internal functions and the outputs in terms of the inputs. This internal description can then be partitioned into subprocesses until the desired level of complexity is reached. Hardware description languages support this design paradigm.

We believe that the adoption of a standard hardware description language in IEEE Standard 1076, specifically VHDL (VHSIC hardware description language) [10], promotes the use of fault avoidance techniques throughout the system design cycle, not just in the design of hardware components, and may decrease generic (design) faults. Descriptive languages

such as VHDL also provide a basis for tools which can aid the specification, design, and validation of both dependable systems and their component building blocks.

After an overview of the design and development process used in current systems, the relevant goals and features of VHDL are be summarized. Next, we address the effect of VHDL on fault avoidance throughout the design process. Following our insights into the potential uses of VHDL in dependable system and component specification, design, and validation, we conclude with a discussion of issues which must be addressed to achieve this potential.

2 Current Design Process

The first step in the typical system design process is a System Requirements and Specifications document (SRS), which defines the boundaries of the service required by the target system, including parameters such as throughput, reliability, maintainability, availability, and the number and types of faults to be tolerated.[2] Using the SRS, the systems designers develop candidate architectures and evaluate them using either their own criteria, or criteria supplied in the SRS. Various high-level system design tools may be used to evaluate key system characteristics of performance, reliability, fault coverage, failure modes, or environmental impact. Any incomplete, impossible, or contradictory specifications in the SRS detected at this stage *should* result in an updated SRS, referred to as SRS'. However, existing erroneous specifications may not be detected at this stage of the design process.

The result of the previous analysis is a set of documents which specifies a design that satisfies the SRS'. This set may include specifications documents for software (SDS) and hardware (HDS) components, which in turn must be evaluated by the hardware and software designers using their own evaluation tools. Similarly, any erroneous specifications in the SDS or HDS documents detected at this time should result in modifications at the appropriate levels to create new documents SRS'', SDS', and HDS'. As before, there is no guarantee that all defects in these documents will be detected, nor that the creation of new versions will not introduce new defects.

Eventually, a set of Design Control Specifications documents (DCS) is produced which detail the programs, components, and ASIC's[3] to be designed. Several stages of testing and redesign may also be necessary throughout this process, requiring several generations of specifications document corrections.

Integration usually occurs after the programs are written, tested, and shown to meet their DCS, and after the ASIC's have been designed, simulated and fabricated. Successful integration may require another round of specifications document corrections, to achieve the "final" versions of SRS^f, SDS^f, HDS^f, and DCS^f for all components.

There is no guarantee that all problems in the specifications at various stages will be discovered, or that the system as integrated achieves the goals of the original system specifier (the customer), without thorough testing. Furthermore, the translation from an idea to natural language[4] design specifications, then to a design, back to natural language hardware and software specifications, or DCS, and then into programs and ASIC's can permit mistakes to creep in due to the inherent difficulties in writing and interpreting natural language text.

3 Features of VHDL

While the most common use of the VHDL design environment has been in design capture and simulation, it also supports the creation of a design from the specification to the completed product. Important features of VHDL are listed in Table 1, presented in detail in [10], and summarized below.

Executable documentation. One of the goals in designing VHDL was to minimize the amount of documentation required to specify and describe the hardware components that were designed, implemented and fabricated. Often, the replacement of a single component can be made impossible, because the precise definition of the behavior of that component is implied by the documentation, not stated explicitly. In fact, VHDL, has been referred to as "executable documentation,"[5] permitting the design description to be used as input to a simulator or to another compatible design tool. In VHDL, the documentation is embedded in the models themselves, in the form of signal, architectural, behavioral, dataflow and structural specifications. Thus, all the information required to understand a component, its behavior, and its interaction with other components is present in the component model.

[2] The design process described is a simplification of techniques presented throughout the literature, and is not taken from any specific system or developer.

[3] ASIC-application specific integrated circuit.

[4] Natural language refers to the spoken or written language, in this case, written English.

[5] p. 46, [11].R

Strong data typing. The data typing used in VHDL requires signals to be of the same type to be combined, and requires input and output ports that are connected either to share a specification or to have an explicit converter in between them. This strong data typing requirement catches many common design mistakes, and forces the development of design rules to ensure that components produced by different vendors or different engineers are compatible.

Top-down specification. The basic component paradigm used in VHDL is the discrete process model shown in Figure 1. The internal parameters of the process description need not be specified, only the outputs generated by specific inputs. The functional details due to a given implementation or technology may be specified later, may be obtained from another vendor or designer, or may never be specified. Such top-down specification is useful in defining how system components interact early in the design, instead of requiring all details to be intact before anything in the system can be simulated. Behavioral, data-flow and structural models are supported, giving the designer latitude in specifying functionality. Thus, the behavior of a component can be specified and simulated, with the gate-level design generated after the inter-component structural information has been verified.

Multi-level simulation. The multi-level simulation supported by VHDL permits the system or component functions to be simulated with the different processes specified at different levels of detail. Both high level and detailed component models can be mixed in the same simulation. Thus, simulation of the model can be done throughout the design process, to ensure that the desired parameters are maintained following each level of expansion. All parts of a given model can be integrated before any hardware is fabricated, eliminating the difficulty faced in testing and integrating simultaneously.

Module reuse. VHDL also supports module reuse, where libraries of components can be maintained based on certain technologies or design rules. The same process model can be simulated using several libraries, even from different vendors. The compilation of libraries of reliable validated components is a key to the development of dependable system building blocks because it permits component reliability to be maximized while minimizing the cost of new systems through module reuse.

- Embedded documentation
- Strong data typing
- Top-down specification
- Multi-level simulation
- Module reuse

Table 1: Important VHDL Features

4 Fault Avoidance and VHDL

Much research has been devoted to the study of how to tolerate faults, with redundancy at the hardware, software, and information levels used to mask, detect, and correct faults. [1] Such redundancy is sufficient to handle restricted fault behaviors, such as single faults that are well separated in time. However, the problem of handling generic or design faults remains, as demonstrated in the following example.

Example 1 Suppose three identical chips perform the same operation, with the results voted to yield a single result. Any fault in a single chip is tolerated, because the results from the remaining good chips will out-vote and mask the bad chip's result. However, the presence of a generic fault in the chip could cause all three chips to fail simultaneously. An incorrect value (or even no value) could be computed, and system failure could result.□

Fault avoidance techniques, such as design rules, careful parts selection, and critical design reviews must be used throughout the design process to minimize the number of specification, implementation and fabrication mistakes which are propagated into the final product. The design strategy supported by VHDL, properly implemented, can provide fault avoidance techniques at all stages in the design of system components. For example, the strong data typing prevents inadvertent connection of ports or components with incompatible signals. Component functions can be described and functional test vectors can be derived at high levels, independent of the technology to be used in their fabrication. The resulting behavioral models can be simulated at different design levels, with multiple levels of simulation in the same design, to continually demonstrate that the design specifications are met. Furthermore, VHDL supports module reuse and the development of a validated library of parts, providing significant cost benefits.

While the need for reliable system components is not new, the use of fault avoidance techniques throughout the component design cycle is required

to ensure the success of future dependable systems. However, VHDL is difficult for many designers to learn, and complex timing behavior cannot be modeled easily in VHDL. Thus, user-friendly VHDL design aids need to be developed and the difficulty in performing timing simulations in VHDL needs to be resolved to achieve this goal.

5 VHDL and the System Design Process

The high level of abstraction and the multiple levels of structural and behavioral specifications supported by VHDL can also provide benefits in specifying, designing, evaluating and validating dependable system designs. However, these benefits require VHDL to be implemented as a descriptive language within an appropriate tool set which includes reliability and performance modeling tools.

5.1 Specification

The ability to specify systems precisely is limited by the lack of precision of natural language text. The designer may not interpret the requirements as the customer meant them, the component functions may not be fully specified, or the specifications taken together may have unforseen effects. The specification of portions of a design using a descriptive language such as VHDL removes the ambiguity inherent in natural language specifications while retaining its specificity. Furthermore, missing specifications may become apparent more quickly as the structure and behavior of components are expanded to different levels.

5.2 Design and Evaluation

The use of VHDL in design synthesis permits important issues, such as protection against different types of faults, to drive the design. Abstract data objects can indicate how faults are handled by various design components. The simultaneous support of top-down and bottom-up design permits existing dependable components to be specified as portions of functions described otherwise only by their inputs and outputs.

Once a VHDL model of the system or component being designed has been written, it can be executed with differing inputs to verify that the system meets its specifications. Faults can be inserted in the model, and the fault protection can be demonstrated. The synthesis and fault-simulation tools currently being developed are aimed at ASIC design and testing, not at the system level. While these tools will also apply at the system level, they will not incorporate the evaluation tools used by the systems designers, such as reliability and performance modeling tools. In a design environment supported by such an integrated tool set, the reliability or performance needed in a component specified by only its inputs and outputs, can be computed based on the required system characteristics and on the characteristics of other components already designed or selected. This gives the designer a guide as to the level of redundancy needed to meet the component characteristics, preventing the expense of over-design and the hazards of under-design.

5.3 Validation

The largest benefit to be achieved by the adoption of VHDL at all levels of system design is in enabling validation of the system by providing traceability to the original specifications document (SRS) in every model component. Since the documentation is contained in each component or subsystem, the adherence to individual specifications is maintained as a part of the system model.

For example, the required fault coverage can be traced at a high level in the design to validate the correspondence of different candidate architectures to the SRS. A simple example is shown below, which employs an enumerated data type to discern between good and bad inputs. This type of validation could be done initially, and repeated as different levels of the design are expanded.

Example 2 A simple VHDL behavioral description of the 3-input voter required by Example 1 demonstrates some features of VHDL at a high level. Using the sample process model depicted in Figure 1, we can define the type of inputs and the type of output expected, and specify the behavior of the voter based on the inputs and other information. The voter model requires three inputs, and produces one output. If any two of the three inputs are good, the output value should be good; otherwise, the output is bad. The enumerated data type {*bad*, *good*} can be used to represent all possible values of the inputs and the output. Then, the voting function can be described by counting the number of *good* entries. If there are two or more good entries, then the output receives the value *good*. Otherwise, the output value is *bad*. VHDL captures the voter behavior and shows that all possible input values, really high-level test vectors, produce the desired output result. Even if the actual values to be voted are bytes of information, requiring more complex comparisons, the coverage of the single input fault can be demonstrated without

designing a byte comparator. A VHDL representation of this example is shown in Figure 2.□

Validation is further aided by the ability to model the integrated system at design time. The integration of different portions of the system before hardware is even built replaces the nightmare of redesigning hardware to adhere to the specifications. Integrated hardware and software models are mapped back to the original SRS, using techniques that can potentially detect design mistakes before they have been implemented in hardware, and perhaps masked by other mistakes.

6 Conclusion

The use of VHDL in hardware component design has the potential to decrease generic faults by permitting the development of verified libraries of dependable components. The adoption of VHDL as a standard, and the multitude of tools being developed and marketed to support ASIC design, such as synthesizers, fault and timing simulators, and design optimizers, will aid in achieving this potential. Graphics packages which automatically generate the VHDL code, and sophisticated debuggers which provide access to all states in a component or chip at a given time will also help to deter and remove design errors. These benefits should be the first to appear.

While the tools and techniques described support system design in VHDL from specification down to gate-level, they will not be sufficient to replace the evaluation and redesign cycles, described in Section 2, used to generate the hardware and software specifications documents. VHDL will need to be implemented within a tool set vastly different from those used in designing and implementing ASIC's. Reliability modeling tools will have to be integrated with VHDL constructs. Tools to estimate the probabilities of faults within different components or subsystems will be needed. This presents a considerable research problem, especially since there is no "standard" reliability modeling tool that works for all architectures.

The potential exists to develop a formal methodology of dependable system design in a VHDL-based environment, using a top-down approach, with a single natural language specifications document. This design process would employ fault avoidance techniques to limit the propagation of design mistakes, while supporting fault tolerance techniques to maintain proper service in the presence of faults due to component wear-out, incorrect use, or environmental factors. However, this potential will not be achieved unless the mature VHDL tools and the formal design techniques needed to produce dependable systems are developed.

```
-Type dual_mode_logic, enumerated data type
—used to discern
—between good and faulty inputs
—can also be converted to another type for
—other simulations
type dual_mode_logic is ('bad','good')

-Entity vote3 is a 3 input voter with a 1ns
—delay which
—tolerates a fault in a single input,
—either an incorrect value or no value
entityvote3 is
      generic( tplh,tphl:time := 1 ns);
      port(a,b,c: in dual_mode_logic;
           d : out dual_mode_logic);
end vote3;

architecture behave of vote3 is
begin
        d <= 'good' after tplh when (a = 'good')
                                    and (b = 'good')
     else
        d <= 'good' after tplh when(a = 'good')
                                    and (c = 'good')
     else
        d <= 'good' after tplh when(b = 'good')
                                    and (c = 'good')
     else
        d <= 'bad' after tphl;
end;
```

Figure 2: Three Input Voter in VHDL

References

[1] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*. Addison-Wesley Publishing Company, 1989.

[2] J. Wensley *et al.*, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proceedings of the IEEE*, vol. 66, October 1978.

[3] J. Goldberg *et al.*, "Development and analysis of the software implemented fault-tolerance (SIFT) computer," NASA contract, Final Report NASA-CR-172146, NASA, February 1984.

[4] J. H. Lala, R. E. Harper, and L. S. Alger, "A design approach for ultrareliable real-time systems," *Computer*, vol. 24, pp. 12–24, May 1991.

[5] C. A. Babikyan, "The fault tolerant parallel processor operating system concepts and performance measurement overview," in *Proceedings, Digital Avionics Systems Conference*, pp. 366–371, IEEE Computer Society, August 1990.

[6] R. E. Harper *Critical Issues in Ultra-Reliable Parallel Processing.* PhD thesis, Massachusetts Institute of Technology, Charles Stark Draper Laboratory, Inc., June 1987.

[7] C. Walter, R. Kieckhafer, and A. Finn, "MAFT: A multicomputer architecture for fault-tolerance in real-time control systems," in *Proceedings, IEEE Real-Time Systems Symposium*, pp. 133–140, IEEE, December 1985.

[8] R. Kieckhafer, C. Walter, A. Finn, and P. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Transactions on Computers*, vol. C-37, pp. 398–405, April 1988.

[9] R. Taylor, P. VanHoff, and C. Walter, "A flexible fault-tolearnt processor for launch vehicle avionics systems," in *Proceedings, 9th Digital AVionics Systems Conference*, pp. 147–152, August 1990.

[10] R. Lipsett, C. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design.* Boston, MS: Kluwer Academic Publishers, first ed., 1989.

[11] L. Gunn, "VHDL: an EDA standard slowly emerges," *Electronic Design*, pp. 45–61, March 1990.

Statement A per telecon
Gary M. Koob ONR/Code 1133
Arlington, VA 22217-5000

NWW 8/22/91